MAIN FRAME PLUS★WARE™ NEWS
A PUBLICATION OF STSC, INC.
April 1986 Issue 4

APL★PLUS® Compiler Released

STSC has announced the release of its compiler for the Mainframe APL★PLUS System. The APL★PLUS Compiler is a tool for speeding up APL programs. It takes standard APL programs as input and produces functionally equivalent machine-language programs that can be run under the Mainframe APL★PLUS System.

Valuable Benefits

APL has always had the advantage of improved programmer productivity, but, in the past, it has often exacted a price in slow execution speed. The compiler changes this, enabling APL to compete better with other languages in the construction of CPU-efficient programs. Faster programs help stretch machine capacity and extend the range of problems that can be practically solved using APL.

Not only can the compiler help programs run faster, but it can save programmer time as well. With the interpreter, it is often a practical necessity to figure out how to write a program without looping, even if unnatural contortions are required to do so. The compiler can make looping algorithms practical and efficient, allowing some programs to be stated in a more forthright and simple manner.

Easy to Use

The compiler is easy to use because it works *with* the interpreter. Programs are developed in the convenient interpreted APL environment. After a program is debugged, a few declarations are added for global objects, and the function is compiled. A compiled function behaves externally like an interpreted function; it can be saved in a workspace, copied from workspace to workspace, or written to a file. The names of compiled functions appear in $\square NL\ 3$ and the *)FNS* display. The $\square CR$ of a compiled function gives the source code of the original APL function. Compiled functions can call interpreted functions and vice versa. When a compiled function encounters an error, it signals an ordinary APL error, not a system error.

Usually, only a few functions in an application (the ones that use the most CPU time) need to be compiled to obtain most of the speedup benefit possible. A monitoring facility ($\square MF$) built into the Mainframe APL★PLUS System makes it easy to identify which functions consume the most CPU time. Early customers have found that speeding up an application using the compiler takes only a small

continued on page 2

Thermo Electron Users Prefer Dynamic, Interactive Capabilities Of STSC's APL★PLUS® System

Thermo Electron Corporation of Waltham, Massachusetts provides industry, utilities, and government with high-technology equipment and services to enhance productivity. We recently talked to Daphne Hatsopoulos, Director of Analysis, about how Thermo Electron has been using the APL★PLUS System in their organization. She reviewed with us the history of the APL★PLUS System at Thermo and how its application has increased productivity in her organization.

Thermo Electron is an international organization with U.S. divisions in Massachusetts, Michigan, Wisconsin, Connecticut, New Mexico, California, and Minnesota, and international divisions in Mexico, Great Britain, and The Netherlands. In 1977, the company was growing rapidly and determined that they needed software to perform corporate consolidation and financial analysis. After an evaluation of programming languages, they chose the APL★PLUS System. The company ran APL on their own IBM 5100 computer for a few years before moving their data to STSC's APL★PLUS Time Sharing System. While on the time sharing system, they wrote customized applications based on STSC's APL★PLUS System. In 1980, Thermo moved their applications and the APL★PLUS System in-house to run on their own IBM 4341.

Initially, the company's needs included monitoring cash flow, investments, budgets, employee benefits, and performing financial modeling. Since writing custom applications for each of these needs was not practical, a challenging technical philosophy was developed with an eye toward the future. If employees had the tools that allowed them to access the necessary corporate financial data, they could write their own applications. The company evaluated programming languages and chose the APL★PLUS System. The system allowed them to develop a framework for accessing corporate data. And, APL's dynamic, interactive capabilities also allowed end users to develop their own tools to solve a myriad of problems.

Three programmers in the company now support, enhance, and maintain two basic pieces of software used by about 40 employees throughout the organization. There are easily 30 different needs being served by the software. A small support staff, well-versed in the software, trains and assists end users in developing their own applications as they solve their own problems.

continued at top of page 6

STSC Signs Acquisition Agreement With Contel

STSC has signed an agreement for the management acquisition of the company from its parent corporation Continental Telecom Inc. (Contel) of Atlanta, GA. Under terms of the agreement, STSC's current officers become majority owners of the company.

At the signing of the agreement, William M. Gibson, STSC President and Chief Executive Officer, said, "Great strides have been made within STSC during the past three years in developing and marketing a number of quality software products. As a result, we are facing the future as a profitable, growing, and vibrant organization. This acquisition reflects our confidence in both STSC and the computer software and services industry in which we compete."

Patricia A. Buteux, Senior Vice President, APL Software Products, added that, "We've also recently combined both our APL mainframe and micro groups into one division. We now have an even stronger team of professionals working together to provide the best possible APL technology and products to help our customers meet their business needs."

In addition to Buteux, the management team for the newly created APL Software Products Division includes J. Stephen Schiavo, Vice President, Sales; Virginia E. Marshalek, Director, APL Marketing; James G. Wheeler, Director, Core Technology Development; Barbara A. Schick, Director, Applications Marketing; and Joyce E. Maker, Manager, New Applications.

APL★PLUS Compiler *continued from page 1*

fraction of the time invested in developing the application. A programmer does not even have to understand how the application works to be able to speed it up.

Proven in the Field

The compiler entered beta test 9 months ago, and the reaction at the test sites has been very positive. Users have compiled a diverse range of programs, providing valuable feedback on how effective the compiler can be in practice.

- *Portfolio Analysis* - A Wall Street financial company using APL for portfolio analysis wanted to reduce internal CPU charges. By compiling 12 of 96 functions, they were able to speed up the application overall by a factor of 2.9. The application took about three years to develop and only two weeks to speed up using the compiler. Faster execution time allows the company to run this application during the day instead of having to wait overnight for results.
- *Trading Strategy Analysis* - Another financial company had an APL program for analyzing stock trading strategies that was expected to consume many hours of CPU time. Compilation sped up their program by a factor of 100 in a benchmark we performed. The same program was submitted to an ace APL programmer for optimization. The programmer spent two days working on it and sped it up by a factor of 3.5, but the client described the resulting program as being "fragile", hard to change, and hard to understand.
- *Manufacturing and Distribution* - At another company, expensive programmer time was being used to speed up manufacturing and distribution applications. Since they got the compiler, four applications have been compiled with speedup factors of 1.2, 1.2, 3, and 9. Programmers at this site say, "We sweated blood the last time we tried to speed up these applications. With the compiler it was much easier."
- *Electronics Design* - An electronics design application used by an aerospace engineering company cost $350,000 a year to run. In a benchmark we performed, compiling 7 out of 125 functions sped up the application overall by a factor of 2. Although the application took several man-years to develop, it was sped up using the compiler in just 34 working hours. The cost of compiling this application can be paid back in five days using the speeded-up application.
- *Structural Analysis* - An engineering consulting firm needed to minimize the time spend by high-priced professionals at terminals. Compiling 6 out of 175 functions in their structural analysis application sped it up by about 25 percent. The savings in CPU charges alone paid for the cost of compilation in just three months using the speeded-up application.
- *Text Retrieval* - A company that provides a text retrieval service was nearing the capacity of their 4341 computer. Using the compiler, their paragraph width adjustment program was speeded up by a factor of 100. This program is part of an application that runs for an elapsed hour per day. This and other speedups provided by the compiler have helped reduce the need for a larger computer. At this site, new projects are planned with the compiler in mind, and programmers are using techniques that were impractical before the compiler was available.

Cost Effective

The APL★PLUS Compiler is a tool heavy APL users cannot affort to be without. For example, if a mainframe computer costs three million dollars, and if one quarter of the machine's capacity is used for APL, the APL portion involves a cost of $750,000. If the compiler, which sells for $30,000, can increase the APL throughput by even one-third, it is making a contribution equivalent to $250,000 of added capacity.

The advent of compiled APL marks a turning point in the development of the language, making it even more productive and practical than before. How much CPU time can you afford to waste needlessly in interpreter overhead? Whether you want to speed up programs to make decisions faster, reduce computing bills, or buy one less mainframe computer this year, you need the APL★PLUS Compiler! To order or obtain additional information about the compiler, contact your STSC representative or call (301) 984-5117.

Meet the Team

In this issue of *Mainframe PLUS★WARE News* we'll introduce the developers of the APL★PLUS Compiler, Clark Wiedmann and Jim Weigang.

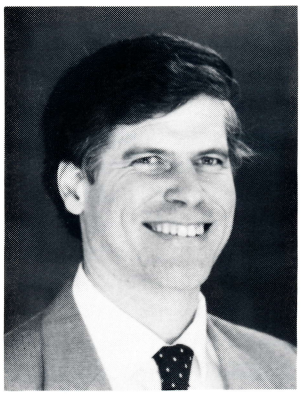
Clark has been with STSC for more than five years and has more than 19 years of programming experience in APL. Clark became familiar with APL while working for the Amherst College Computing Center. Since then, he has participated in the construction of two APL interpreters (for CDC mainframe computers) and has led the construction of a large data analysis package written in APL. Clark founded the ANSI APL Standards Committee in 1979 and served as its first chairman. He has written a book on APL programming, an APL reference manual, and numerous articles, and he served as the language editor of *APL Quote Quad* for three years.

Jim has been with STSC for two years and has more than nine years of programming experience in APL. Jim learned APL in a mathematics course at the University of Massachusetts, and shortly thereafter was hired as a part-time programmer working with Clark on IDAP, the Interactive Data Analysis Package. Jim has earned a living as an APL programmer ever since, writing many APL applications including graphics packages, text editors, an image processing system, and a successor to IDAP. He has taught APL at the University of Massachusetts and Hampshire College, and has written an introductory text on APL and several articles and manuals.

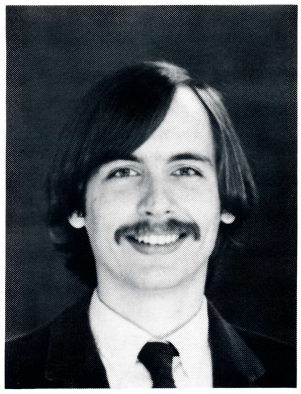
The APL★PLUS Compiler has certainly been Clark's most ambitious project to date; he has been working on it for most of the past decade. Jim has been working with Clark on the compiler for the past two years.

Both Clark and Jim are hard-core APL programmers who say that they like the language because they can get so much more done using APL. Clark points out that, although APL has always been more productive for programmers, interpreted APL lacks the pizazz of CPU-efficient programs. Jim expresses the hope that the compiler will make APL useful in new areas and will open the eyes of APL programmers to new algorithms and techniques from computer science that are not practical with interpreters.

Clark attributes his success in writing the compiler to analyzing the APL efficiency problem carefully and then learning about conventional compilers. Historically, there has been very little overlap between those who know APL and those who know the techniques used in optimizing compilers. Knowing about both enabled Clark to bring proven compiler technology to bear on the APL performance problem. He saw problems with using optimization techniques commonly discussed by APL implementors (such as "dragging and beating") that fail to provide speedup benefit unless arrays are large. The interpreter already processes large arrays fairly efficiently; it is the processing of small arrays that needs help.



Clark Wiedmann



Jim Weigang

Clark's and Jim's future plans include speedups and improvements to the compiler and compiled functions, and other work to improve the quality and usefulness of the Mainframe APL★PLUS System.

Clark is married, has two children, and lives in a house he designed and built himself in Ashfield, Massachusetts. Jim is single and lives nearby in Northampton. They both work at STSC's Haydenville, Massachusetts office, which is located in the Brassworks, a historic converted brass mill.

New Release Provides Many Improvements to APL★PLUS® Mainframe System

A new release (Version 5.1) of the APL★PLUS Mainframe System is now available for both MVS and VM. It incorporates many improvements and bug fixes, including:

Deorphaner and Reorganizer for SHAREFILE™

Two new SHAREFILE utilities provide a convenient mechanism for reclaiming unusable space in the file system. The deorphaner recovers space lost when the operating system is halted without first shutting down the SHAREFILE System. The reorganizer packs files to recover space lost due to □*FREPLACE* operations. Together, these utilities save operator time by making it unnecessary to restore the file system from tape just to reclaim lost space.

Support for GDDM Releases 3 and 4

The APL★PLUS System and AP126, the graphical display auxiliary processor, now support releases 3 and 4 of GDDM. This improvement includes support for 3179-type terminals.

Faster Tape Utilities on VM

The SHAREFILE restore, retrieve, and selective dump utilities have been sped up in the VM version of the APL★PLUS System. The restore utility, for example, is up to 10 times faster than before. The speedup was obtained by passing data a tape block at a time rather than a file component at a time. An additional benefit of this change is that the tape utility operations no longer require a virtual machine size capable of holding the largest component. The MVS version of the APL★PLUS System has used this technique for some time.

Improved □MF Resolution on MVS

On the MVS version of the APL★PLUS System, □*MF* now has greater resolution and produces more accurate results. Previously, the CPU time counter used by □*MF* was updated only when a user's time slice ended, causing random clumping of execution times. Now, a counter update

continued on page 6

Using Linked Lists

by Jim Weigang

One common task in programming is having to process different groups of data one group at a time in a loop. The way APL programmers typically solve this problem results in programs that can be critically inefficient when processing large amounts of data. By using subroutines in the FASTFNS workspace to manage linked lists, execution time can be reduced dramatically for some important cases.

For example, assume that data is stored in a 2-column matrix with the first column holding the grouping variable and the second holding the data variable. Such a data matrix might look like this:

Group	*Data*
2	185
3	71
1	102
2	75
1	51
3	22

Typically, APL programmers use compression to select the members of a group. A loop that processes the data one group at a time might look something like this:

```
 GROUP←MAT[;1]
 DATA←MAT[;2]
 NGRPS←⌈/GROUP
 I←0
L1:→(NGRPS<I←I+1)/L2
 D←(GROUP=I)/DATA
 ⍝ ...PROCESS D ...
 →L1
L2:
```

The problem with this approach is that it has to process *all* the data to extract a single group. If the number of different groups is proportional to the amount of data present (say, *NGRPS*=*M*×⍴*DATA*), the selection process alone processes *M*×(⍴*DATA*)*2 elements, resulting in a O(N*2) (read "big-oh N-squared") program where N is ⍴*DATA*. [See the December 1985 issue of *Mainframe PLUS★WARE News* for more information about big-oh notation.]

By using linked lists, the members of a group can be retrieved without having to process all the data. Unnecessary searching is avoided by chaining together the members of each group so they can be found directly, without having to search for them. The ⎕*MAKELIST* function forms a linked list from a data matrix, and the ⎕*RETRIEVE* function returns data elements belonging to a particular group. Using these functions, the loop above can be restated as:

```
 NGRPS←⌈/MAT[;1]
 ΔLIST←NGRPS ⎕MAKELIST MAT
 I←0
L1:→(NGRPS<I←I+1)/L2
 D←ΔLIST ⎕RETRIEVE I
 ⍝ ...PROCESS D...
 →L1
L2:
```

Here, variable Δ*LIST* holds the linked list, which is an ordinary integer vector holding a special data structure used by the linked list subroutines.

To estimate the savings in the selection process possible by using ⎕*MAKELIST* and ⎕*RETRIEVE*, the program fragments above were timed for various amounts of data (where N is the number of data elements):

	CPU time, secs.	
N	*Compress*	⎕*RETRIEVE*
10	.0014	.0026
100	.0189	.0183
1000	1.3284	.1753
10000	126.8112	1.7717

Although using linked lists is slightly less efficient for small arrays, the savings in CPU time when processing large amounts of data is enormous.

Extending a List

In situations where a dynamic data set is stored in a linked list, it may be necessary to extend the list. The ⎕*ADDLIST* function can be used to do this. ⎕*ADDLIST* takes a two-column matrix of group and data variables as its argument and adds it to the linked list in global variable Δ*LIST*. (A global variable is used because the list is both an argument and a result.)

Using the matrix approach, a list can be extended using catenation, as in:

```
MAT←MAT,[1]NEWDATA
```

Using linked lists, this statement can be replaced with:

```
⎕ADDLIST NEWDATA
```

When adding large amounts of data a little at a time, ⎕*ADDLIST* can be significantly more efficient than catenation. Each catenation requires copying all the old data to a new, slightly larger container. If a matrix is built one row at a time using catenate, the execution time is O(N*2). ⎕*ADDLIST* avoids this inefficiency by padding the linked list by 50 percent more length than required when it must be made larger. As a result, building a list with ⎕*ADDLIST* takes O(N×log N) execution time. However, note that catenate is a very fast O(N*2) and beats ⎕*ADDLIST* until arrays are rather large. The principal advantage of ⎕*ADDLIST* is that it saves you from having to use ⎕*MAKELIST* every time you add a new piece of data.

continued on page 6

MAIN FRAME PLUS★WARE™ NEWS
A PUBLICATION OF STSC, INC.

Mainframe PLUS★WARE News is published by STSC, Inc. and is distributed free of charge to the company's customers. Comments regarding the newsletter should be sent to:

Christopher Bigler, Editor
STSC, Inc.
2115 East Jefferson St.
Rockville, MD 20852

©1986 STSC, Inc. All rights reserved. Printed in U.S.A.

APL★PLUS is a registered trademark of STSC, Inc.
PLUS★WARE and SHAREFILE are trademarks of STSC, Inc.

STSC's APL Training Courses Help Provide Solutions to Daily Business Problems

STSC kicks off its new Educational Services Program in 1986 with a comprehensive series of APL training courses designed to help participants more effectively use APL's power to solve daily business problems.

The year-round schedule of courses, to be held at STSC's offices in Rockville, Maryland and in New York City, includes APL training for beginning, intermediate, and advanced users. An $800 fee for the four-day courses covers all curriculum materials, lunch, and refreshments.

According to Tama Traberman, APL Educational Services Manager, the APL training program features new curricula that incorporates pertinent business and professional applications into the language-learning environment.

"Rather than teach APL in an isolated fashion," Traberman said, "we will help participants quickly develop a high level of skills that will allow them to apply APL to their business problems from the very first lesson."

The curriculum for the introductory APL courses covers primitive functions, operators, logical relationships, function definition, files, formatting, and all the STSC enhancements to its APL products.

The intermediate APL courses will focus on advanced primitive APL functions, nested and heterogeneous arrays, idioms, and other tools.

A series of special topic courses scheduled for April 14-17 in Rockville and May 12-15 in New York City are designed to teach participants how to use APL to enhance and improve the systems they currently use. Participants will learn a wide range of trouble-shooting skills that will enable them to analyze problems and devise fixes for the system. The courses will teach participants to build tools for maintaining and maximizing the operations of their systems.

Maximum class size is 12, so be sure to register early to reserve space. To register, send in the form below, or call (301) 984-5490 for the Rockville classes and (212) 751-9305 for the New York City classes. Registration forms must be accompanied by a check or purchase order one week prior to class starting date. Telephone registrations require a purchase order number. Please note there is a $10 nonrefundable enrollment fee.

1986 APL Class Schedule

(All classes introductory unless noted otherwise)

Rockville	**New York City**
April 14-17 (Special Topics)	April 7-10
May 19-22 (Intermediate APL)	May 12-15 (Special Topics)
June 16-19	June 2-5 (Intermediate APL)
July 14-17	July 21-24
August 18-21	August 4-7
September 22-25	September 8-11
October 10-13	October 6-9
November 10-13	November 17-20
December 8-11	December 1-4

REGISTRATION FORM

STSC

Mail this registration form with payment or valid purchase order to the APL Seminar Coordinator in the city where the class will be held. Submit a separate form for each person attending.

Rockville Classes:
2115 East Jefferson St.
Rockville, MD 20852

New York Classes:
747 Third Avenue
New York, NY 10017

Name_____ Title_____

Organization _____

Address _____

City_____ State_____ ZIP_____

Daytime Telephone _____

Name of Class_____ Date_____

☐ Enclosed is a check for $_____ for _____ attendee(s).

Please bill my company. P.O. Number is _____.

Billing address_____

City_____ State_____ ZIP_____

☐ I cannot attend at this time but please add my name to your mailing list for information on future classes.

Thermo Electron *continued from page 1*

The two basic pieces of software developed by the company's programmers are CUBE, a three-dimensional spreadsheet package, and TRACK, a database management system.

CUBE is tied to Thermo's corporate financial database and allows users to do modeling, custom report writing, and graphics. The software will typically handle dimensions such as time, budget, and line items for corporate consolidation purposes. However, users can define dimensions as anything they want them to be by using simple English-like commands. The report writer allows analysts to define where the data should come from, which items they want to see for specific time periods, and the kinds of basic calculations they want. The function SUM, for example, is an intelligent program which looks at data elements and adds only like items or, when appropriate, adds only quarterly totals when looking at yearly financial data.

Security and data integrity are major concerns at Thermo since 30 to 40 employees need to access the company's financial data. Security is handled by built-in features provided with STSC's SHAREFILE System. SHAREFILE gives the database administrator the ability to specifically assign read or write access to data files. Thermo uses this capability to restrict user access to specific fields of their financial database.

To round out the functionality of CUBE, Thermo has built a bridge between the APL★PLUS System and a non-APL, microcomputer-based graphics tool which allows users to develop color graphic representations of their data for presentation-quality reports.

The second piece of software developed by Thermo with the APL★PLUS System is called TRACK. TRACK is a record-oriented database management system. The company uses TRACK throughout the organization to perform library catalog management, to maintain mailing lists and stockholder information, and for personnel record management. TRACK's master personnel file contains employee benefit, pension, and stock purchase information, in addition to standard employee data. TRACK is used to automatically generate letters to employees informing them of changes in company benefits or stock plan information. The system allows the user to add records and select data based on specific criteria like an employee's start date or length of employment with the company. A new user can be trained and actively working with TRACK in just a few hours.

Thermo Electron will continue to enhance TRACK and CUBE. TRACK will soon include standard report options as well as the ability to save selection criteria.

Daphne Hatsopoulos feels that the APL★PLUS System has made Thermo's end users much more demanding about their software tools. "At one point," she explains, "a mainframe accounting system was brought into Thermo and discarded by end users in favor of the dynamic, interactive capabilities of the APL★PLUS System." At Thermo Electron, users are not at the mercy of the software but are making the software work for them.

Linked Lists *continued from page 4*

Other Kinds of Data

As written, *⍁MAKELIST*, *⍁RETRIEVE*, and *⍁ADDLIST* are designed to work with integer data consisting of one grouping and one data variable. What if your data consists of character strings or real numbers? And what if you have more than one grouping variable?

If your data is not integers, it can be stored in a separate array, and the result of *⍁RETRIEVE* can be used to index the data in the array. If variable *DATAMAT* is a matrix of data, a linked list can be built with:

```
M←GROUP,[1.5]⍳1↑⍴DATAMAT
ΔLIST←NGRPS ⍁MAKELIST M
```

Data can be retrieved with:

```
D←DATAMAT[ΔLIST ⍁RETRIEVE I;]
```

This technique preserves the speed advantage of *⍁RETRIEVE* because selected data elements are retrieved without having to process all the data.

If you have more than one grouping variable, the *RANKMAT* function can be used to map the grouping variables onto a single ordinal grouping variable suitable for use with *⍁MAKELIST*. If the grouping variables are stored in the columns of matrix *GROUPMAT*, the linked list can be built with:

```
GROUP←RANKMAT GROUPMAT
NGRPS←⌈/GROUP
M←GROUP,[1.5]DATA
ΔLIST←NGRPS ⍁MAKELIST M
```

Where Are They?

The functions *⍁MAKELIST*, *⍁RETRIEVE*, *⍁ADDLIST*, and *RANKMAT* can be found in the FASTFNS workspace in library 3. These functions are written in TABL, STSC's proprietary systems programming language. Documentation for these and other precompiled utility functions is available both in the workspace and in the *FASTFNS Reference Manual*.

New Release *continued from page 3*

is forced whenever the value is read, providing resolution equal to that on VM. This change also improves the accuracy of *□AI*[2] and the *)AI* command of AP100.

Other Improvements and Bug Fixes

More than a dozen new precompiled utility functions have been added to the FASTFNS workspace. *□DEF* and *□FX* have been changed so they may now replace compiled functions. Two error messages signalled by compiled functions have been made clearer.

Bugs have been fixed in indexing, dyadic iota, equivalent, *□DEF*, *□FX*, *□EA*, *□MF*, the Compiler Support Package, XEDIT support, the Session Manager, the VSAM processor (AP123), the TSO command processor (AP100), and the MFFNS and OPFNS workspaces.

These changes reflect STSC's commitment to improving the APL★PLUS Mainframe System and serving its users. Expect to see further improvements to the SHAREFILE System and innovative new productivity enhancements in future releases.